REMARKS

Claims 1-20 are pending in the current application and stand rejected. For the following reasons, applicant respectfully traverses the rejection and requests that the final rejection be withdrawn and the application passed to allowance.

Applicant has amended claims 2 and 4 to overcome the rejections thereto under 35 U.S.C. § 112, second paragraph, set forth in paragraph 5 of the Office Action. With respect to the term "malicious" in claim 16, applicant respectfully traverses the rejection. Applicant respectfully asserts that the term "malicious" is well known in the computer arts. For example, as described in paragraph [0028] of the present specification, one example of malicious behavior in the computer arts is "do[ing] something contrary to an information system security policy." Similarly, with respect to claim 18, applicant respectfully asserts that the term "damaging" is well known in the computer arts. Examples of "damaging system resources," without limitation, include rendering system resources unusable, less efficient than their specifications for normal operation, or causing them to malfunction in some manner. Further, applicant respectfully asserts that the term "elevating" in the context of the computer arts is well known. In one context for example, the term elevating means "raising" or "expanding." For example, "elevating user privileges" refers to raising the privilege level. By "elevating user privileges" for example, an intruder to a computer system may be able to access computer resources that he or she would not be able to access with lower user privileges.

Claims 1-20 have been rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. Pub. 2003/0121027 to Hines et. al. ("Hines"). Hines is directed to a system and method for debugging concurrent systems designs. (Hines at [0032-33].) Because

concurrent designs have no distinct meaning for an instant in execution time, Hines is a debugger that operates on a principal of distributed event-based debugging rather than state-based debugging. (Hines at [0477-0480].)

Applicant respectfully asserts that Hines does not disclose several of the elements recited in claim 1. First, debuggers, such as the system disclosed in Hines, do not generate a normal execution trace for a software program as required by the first element of claim 1. Rather, they proceed through a program's execution until they reach particular breakpoints (pre-established by a developer using the debugger) so that the developer can investigate the state of program execution to that point. (Hines at [0477]). the debugger disclosed Hines is no different. Though somewhat more complex because of its distribution event-based methodology, the debugging tool disclosed in Hines is still limited to allowing developers to establish future breakpoints wherein all events that the future breakpoints depend upon have already occurred. (Hines at [0488-490].) In operation, the debugging tool disclosed in Hines moves the future event breakpoint to the past, prior to allowing the developer to perform any debugging tasks. (Hines at [0489].) However, there is no disclosed "generating a normal execution trace for the software program" as recited in claim 1.

Assuming *arguendo* that Hines does disclose "generating a normal execution trace for the software program," which it does not, applicant respectfully asserts that Hines does not disclose "applying a learning algorithm to the normal execution trace to build a finite automaton." The Office Action states that "a debugger is a learning algorithm used to build a finite automaton ... an operating program." The Office Action further explains on page 8, that "a debugger represents a learning process or algorithm,

related to anomaly detection to identify the reasons why a software system or part thereof fails to function properly .... anomaly detection." Applicant respectfully traverses the official notice of the term "debugger" provided in the Office Action.

Applicant is unaware of any common knowledge in the art that a debugger is used as a learning algorithm to build a finite automaton or that a debugger can identify the reasons why a software program fails to function properly. Rather, applicant respectfully asserts that a debugger is software used by a programmer to study a program's behavior using breakpoints to stop the program and display a program state. It is a tool for the programmer to determine why a software system or part thereof fails to function properly, but it does not itself identify the reasons for the software failure. Indeed, such analysis could thwart the very purpose of a debugger. It is not supposed to make assessments of the software. Rather, it is simply to tell the programmer what is happening at various points in the execution.

This understanding is consistent with the definition of the term "debugger" found, for example, in ALAN FREEDMAN, THE COMPUTER GLOSSARY 100 (8th ed. 1998) (copy submitted herewith). Given applicant's traversal of the official notice provided for the meaning imparted to the term debugger in the Office Action, should the Examiner maintain his rejection on this basis, applicant respectfully requests that the Examiner support his interpretation of the term debugger as a learning algorithm with adequate evidence as required by M.P.E.P. § 2144.03, or withdraw the rejection.

Even, assuming *arguendo*, a debugger is a learning algorithm, which it is not, applicant respectfully asserts the debugger of Hines cannot be the "learning algorithm" recited in claim 1. The "learning algorithm" recited in claim 1 is used "to build a finite

automaton." That is, until application of the "learning algorithm" in the present invention as recited in claim 1, there is no finite automaton. This is completely different from Hines. In Hines a debugger is used to study operation of a concurrent system. To the extent that concurrent system can be represented by a finite automaton, the debugger does not build the system. The system is *already built*. Hines' debugger is merely used to study that system. Consequently, to the extent Hines' debugger is a learning algorithm, and the concurrent system being studied represents a finite automaton (which they are not), at best Hines discloses that the "learning algorithm" (Hines' debugger) is applied to a *pre-existing* "finite automaton" (the concurrent system under study). As a result, the "learning algorithm" in Hines cannot be used to build the finite automaton as recited in claim 1. Consequently, applicant respectfully requests that the Examiner reconsider and withdraw the rejection.

Applicant respectfully asserts that Hines further does not disclose "applying an examination algorithm to the finite automaton to identify undesirable transition states in the finite automaton and to create a labeled finite automaton." The Office Action asserts on pages 3 and 7-8 that determination of "consistent cuts" satisfies this element of the claims. Applicant respectfully disagrees. As described above, "consistent cuts" as defined in Hines are breakpoints that can be used by the disclosed debugger in a concurrent system, which requires causality be maintained on a global scale. (Hines at 480-81.) Consequently, the consistent cuts are used by Hines' debugger as global breakpoints in the concurrent system under study.

As described in Hines, "FIG. 39 shows that consistent cuts can be represented as a jagged line across the space/time diagram ...." On page 8, the Office Action asserts that

"[f]igure 39 would be part of a labeled finite automaton." Applicant understands that the Office Action refers to the space/time diagram illustrated in figure 39 as the asserted "part of a *labeled* finite automaton." (Emphasis supplied.) While applicant does not believe Hines discloses an "examination algorithm" as recited in claim 1, assuming *arguendo* that Hines does disclose an "examination algorithm," figure 39 fairly stands only for disclosing "applying an examination algorithm to a *labeled finite automaton* to identify undesirable transition states ...." That is, figure 39 does not disclose the recited "applying an examination algorithm to a *finite automaton* to identify undesirable transition states...."

Moreover, to the extent Hines can be said to disclose an examination algorithm at all, that examination algorithm cannot "create a labeled finite automaton" as recited in claim 1. This is because any analysis of the state/space diagram of figure 39 must be performed after it has been created. That is, to determine consistent cuts as shown in figure 39, the state/space diagram must already exist. As a result, any algorithm to examine figure the state/space diagram of figure 39 cannot at the same time create it. Accordingly, applicant respectfully requests that the Examiner reconsider and withdraw the rejection of claim 1.

Moreover, if figure 39 is "part of the labeled finite automaton" as asserted in the Office Action, applicant respectfully asserts that Hines cannot disclose "applying the labeled finite automaton to an execution trace associated with the executing software program." As described in Hines, the state/space diagram of figure 39 is used to determine consistent cuts that can be used as breakpoints in the debugging process. There is no description of applying the state/space diagram to an execution trace

associated with the executing software. Thus, Hines does not disclose "applying the labeled finite automaton to an execution trace associated with the executing software program" as recited in claim 1. Accordingly, applicant respectfully requests the Examiner to reconsider and withdraw the rejection of claim 1.

In summary, applicant respectfully asserts that for all of the reasons provided above, Hines does not disclose each and every element of claim 1 of the present application. Accordingly, applicant respectfully requests that the Examiner reconsider and withdraw the rejection of claim 1, and its dependent claims 2-20, as being anticipated by Hines.

In view of the foregoing all of the claims in this case are believed to be in condition for allowance. Should the Examiner have any questions or determine that any further action is desirable to place this application in even better condition for issue, the Examiner is encouraged to telephone applicants' undersigned representative at the number listed below.

PILLSBURY WINTHROP SHAW PITTMAN LLP
P.O. Box 10500
McLean, VA 22102
Tel: (703) 770-7900

Respectfully submitted,

Date: January 17, 2006

By: David C. Isaacson
Registration No. 38,500

DCI/ar

Customer No. 00909